The Complexity of Valued Constraint Satisfaction

Peter Jeavons* Andrei Krokhin† Stanislav Živný‡

Abstract

We survey recent results on the broad family of problems that can be cast as valued constraint satisfaction problems. We discuss general methods for analysing the complexity of such problems, give examples of tractable cases, and identify general features of the complexity landscape.

1 Introduction

Computational problems from many different areas involve finding an assignment of values to a set of variables, where that assignment must satisfy some specified feasibility conditions and optimise some specified objective function. In many such problems the objective function can be represented as a sum of functions, each of which depends on some subset of the variables. Examples include: Gibbs energy minimisation, Markov Random Fields (MRF), Conditional Random Fields (CRF), Min-Sum Problems, Minimum Cost Homomorphism, Constraint Optimisation Problems (COP) and Valued Constraint Satisfaction Problems (VCSP) [6, 23, 68, 85, 87, 89].

We focus in this article on a generic framework for such problems that captures their general form. Bringing all such problems into a common framework draws attention to common aspects that they all share, and allows a very general algebraic approach for analysing their complexity to be developed. The primary motivation for this line of research is to understand the general picture of complexity within this general framework, rather than to develop specialised techniques for specific applications. We will give an overview of this algebraic approach, and the results that have been obtained by using it.

*Department of Computer Science, University of Oxford, Peter.Jeavons@cs.ox.ac.uk

†School of Engineering and Computing Sciences, University of Durham, Andrei.Krokhin@durham.ac.uk (Andrei Krokhin is supported by the UK EPSRC grant EP/H000666/1)

‡Department of Computer Science, University of Oxford, standa@cs.ox.ac.uk
(Stanislav Živný is supported by a Royal Society University Research Fellowship)

The generic framework we use is the *valued constraint satisfaction problem* (VCSP), defined formally as follows. Throughout the paper, let D be a fixed finite set and let $\overline{\mathbb{Q}} = \mathbb{Q} \cup \{\infty\}$ denote the set of rational numbers with (positive) infinity.

Definition 1. We denote the set of all functions $\phi : D^m \to \overline{\mathbb{Q}}$ by $\mathbf{\Phi}_D^{(m)}$ and let $\mathbf{\Phi}_D = \bigcup_{m \geq 1} \mathbf{\Phi}_D^{(m)}$. We will often call the functions in $\mathbf{\Phi}_D$ *cost functions* over D.

Let $V = \{x_1, \ldots, x_n\}$ be a set of variables. A *valued constraint* over V is an expression of the form $\phi(\mathbf{x})$ where $\mathbf{x} \in V^m$ and $\phi \in \mathbf{\Phi}_D^{(m)}$. The number m is called the *arity* of the constraint, the function ϕ is called the *constraint function*, and the tuple $\mathbf{x}$ the *scope* of the constraint.

We will call the elements of D *labels* (for variables), and say that the cost functions in $\mathbf{\Phi}_D$ take *values*.

Definition 2. An instance of the *valued constraint satisfaction problem* (VCSP) is specified by a finite set $V = \{x_1, \ldots, x_n\}$ of variables, a finite set D of labels, and an *objective function* Φ expressed as follows:

$$\Phi(x_1, \ldots, x_n) = \sum_{i=1}^{q} \phi_i(\mathbf{x}_i) \tag{1}$$

where each $\phi_i(\mathbf{x}_i)$, $1 \leq i \leq q$, is a valued constraint over V. Each constraint can appear multiple times in Φ.

The goal is to find an *assignment* of labels to the variables (or *labelling*) that minimises Φ.

Note that the value of the function Φ for any assignment of labels to the variables in V is given by the sum of the values taken by the constraints; this value will sometimes be called the *cost* of the assignment. An infinite value for any constraint indicates an infeasible assignment.

If the constraint functions in some VCSP instance are finite-valued, i.e., take only finite values, then every assignment is feasible, and the problem is to identify an assignment with minimum possible cost (i.e., we need to deal only with the optimisation issue). On the other hand, if each constraint function in an instance takes only two values: one finite value (possibly specific to the constraint) and ∞, then all feasible assignments are equally good, and so the only question is whether any such assignment exists (i.e., we need to deal only with the feasibility issue). If we have neither of the above cases then we need to deal with both feasibility and optimisation.

In Section 2 we give examples to show that many standard combinatorial optimisation problems can be conveniently expressed in the VCSP framework. In Section 3 we define certain algebraic properties of the constraints that can be used

to identify many tractable cases. Section 4 describes the basics of a recently developed general algebraic theory for analysing the complexity of different forms of valued constraints. In Section 5 we use this algebraic theory to identify several tractable and intractable cases, and in Section 6 we discuss approximation. In Section 7 we discuss the oracle model for representing the objective function. Finally, Section 8 gives a brief summary and identifies some open problems.

2 Problems and frameworks captured by the VCSP

In this section we will give examples of specific problems and previously studied frameworks that can be expressed as VCSPs with restricted forms of constraints.

Definition 3. Any set $\Gamma \subseteq \mathbf{\Phi}_D$ is called a *valued constraint language* over D, or simply a *language*. We will denote by VCSP(Γ) the class of all VCSP instances in which the constraint functions are all contained in Γ.

Valued constraint languages may be infinite, but it will be convenient to follow [11, 17] and define the complexity of a valued constraint language in terms of its finite subsets. We assume throughout that P≠NP.

Definition 4. A valued constraint language Γ is called *tractable* if VCSP(Γ') can be solved (to optimality) in polynomial time for every finite subset $\Gamma' \subseteq \Gamma$, and Γ is called *intractable* if VCSP(Γ) is NP-hard for some finite $\Gamma' \subseteq \Gamma$.

One advantage of defining tractability in terms of finite subsets is that the tractability of a valued constraint language is independent of whether the cost functions are represented explicitly (say, via full tables of values, or via tables for the finite-valued parts) or implicitly (via oracles).

Example 5 (NAE-SAT)**.** Let $D = \{0, 1\}$ and let Γ_{nae} be the language that contains just the single ternary cost function $\phi_{\mathsf{nae}} : D^3 \to \overline{\mathbb{Q}}$ defined by

$$\phi_{\mathsf{nae}}(x, y, z) \stackrel{\text{def}}{=} \begin{cases} \infty & \text{if } x = y = z \\ 0 & \text{otherwise} \end{cases}.$$

The problem VCSP(Γ_{nae}) is exactly the Not-All-Equal Satisfiability problem, also known as the 3-Uniform Hypergraph 2-Colourability problem. This problem is NP-hard [33], so Γ_{nae} is intractable.

Example 6 (Max-k-Cut)**.** Let Γ_{xor} be the language that contains just the single binary cost function $\phi_{\mathsf{xor}} : D^2 \to \overline{\mathbb{Q}}$ defined by

$$\phi_{\mathsf{xor}}(x, y) \stackrel{\text{def}}{=} \begin{cases} 1 & \text{if } x = y \\ 0 & \text{if } x \neq y \end{cases}.$$

The problem VCSP(Γ_{xor}) corresponds to the problem of minimising the number of monochrome edges in a k-colouring (where $k = |D|$) of the graph G formed by the scopes of the constraints. This problem is known as the Maximum k-Cut problem (or simply Max-Cut when $|D| = 2$), and is NP-hard [33].

Hence, for any choice of D, the language Γ_{xor} is intractable.

Example 7 (Potts model)**.** Let Γ_{Potts} be the language that contains all unary cost functions and the single binary cost function $\phi_{\mathsf{Potts}}\colon D^2 \to \overline{\mathbb{Q}}$ defined by

$$\phi_{\mathsf{Potts}}(x,y) \stackrel{\text{def}}{=} \begin{cases} 0 & \text{if } x = y \\ 1 & \text{if } x \neq y \end{cases}.$$

The problem VCSP(Γ_{Potts}) corresponds to finding the minimum energy state of the Potts model from statistical mechanics (with external field) [72]. This model is also used as the basis for a standard Markov Random Field approach to a wide variety of problems in machine vision [6]. For $|D| = 2$, the function ϕ_{Potts} is submodular (see Example 18) and we will show that this implies that Γ_{Potts} is tractable. For $|D| > 2$, Γ_{Potts} is intractable as it includes, as a special case, the multiway cut problem, which is NP-hard [27].

Example 8 ((s,t)-Min-Cut)**.** Let $G = (V, E)$ be a directed weighted graph such that for every $(u,v) \in E$ there is a weight $w(u,v) \in \mathbb{Q}_{\geq 0}$ and let $s, t \in V$ be distinguished source and target nodes. Recall that an (s,t)-*cut* C is a subset of vertices V such that $s \in C$ but $t \notin C$. The weight, or the size, of an (s,t)-cut C is defined as $\sum_{(u,v)\in E, u\in C, v\notin C} w(u,v)$. The (s,t)-Min-Cut problem consists in finding a minimum-weight (s,t)-cut in G. We can formulate the search for a minimum-weight (s,t)-cut in G as a VCSP instance as follows.

Let $D = \{0,1\}$. For any label $d \in D$ and cost $c \in \overline{\mathbb{Q}}$, we define

$$\eta_d^c(x) \stackrel{\text{def}}{=} \begin{cases} 0 & \text{if } x = d \\ c & \text{if } x \neq d \end{cases}.$$

For any weight $w \in \mathbb{Q}_{\geq 0}$, we define

$$\phi_{\mathsf{cut}}^w(x,y) \stackrel{\text{def}}{=} \begin{cases} w & \text{if } x = 0 \text{ and } y = 1 \\ 0 & \text{otherwise} \end{cases}.$$

We denote by Γ_{cut} the set $\{\eta_0^\infty, \eta_1^\infty\} \cup \{\phi_{\mathsf{cut}}^w \mid w \in \mathbb{Q}_{\geq 0}\}$. A minimum-weight (s,t)-cut in a graph G with set of nodes $V = \{x_1, \ldots, x_n\}$ corresponds to the set of variables assigned the label 0 in a minimal cost assignment to the VCSP instance defined by

$$\Phi(x_1, \ldots, x_n) \stackrel{\text{def}}{=} \eta_0^\infty(s) + \eta_1^\infty(t) + \sum_{(x_i,x_j)\in E} \phi_{\mathsf{cut}}^{w(x_i,x_j)}(x_i, x_j).$$

The unary constraints ensure that the source and target nodes must be assigned the labels 0 and 1, respectively, in any minimal cost assignment.

Furthermore, it is an easy exercise to show that any instance of VCSP(Γ_{cut}) on n variables can be solved in $O(n^3)$ time by a reduction to (s,t)-Min-Cut and then using the standard algorithm [35]. Hence Γ_{cut} is tractable.

Example 9 (Minimum Vertex Cover)**.** The Minimum Vertex Cover problem asks for a minimum size set W of vertices in a given graph $G = (V, E)$ such that each edge in E has at least one endpoint in W. This problem is NP-hard [33].

Let $D = \{0, 1\}$. We define

$$\phi_{\mathsf{vc}}(x, y) \stackrel{\text{def}}{=} \begin{cases} \infty & \text{if } x = y = 0 \\ 0 & \text{otherwise} \end{cases}.$$

We denote by Γ_{vc} the language $\{\phi_{\mathsf{vc}}, \eta_0^1\}$, where η_0^1 is the function defined in Example 8 that imposes unit cost for any variable assigned the label 1. A minimum vertex cover in a graph G with set of vertices $V = \{x_1, \ldots, x_n\}$ corresponds to the set of vertices assigned the label 1 in some minimum cost assignment to the VCSP(Γ_{vc}) instance defined by

$$\Phi(x_1, \ldots, x_n) \stackrel{\text{def}}{=} \sum_{x_i \in V} \eta_0^1(x_i) + \sum_{(x_i, x_j) \in E} \phi_{\mathsf{vc}}(x_i, x_j).$$

The binary constraints ensure that in any minimal cost assignment at least one endpoint of each edge belongs to the vertex cover.

Furthermore, it is easy to convert any instance of VCSP(Γ_{vc}) to an equivalent instance of Minimum Vertex Cover by repeatedly assigning the label 1 to all variables which do not appear in the scope of any unary constraints and removing these variables and all constraints involving them. Hence Γ_{vc} is intractable.

We will now show how several broad frameworks previously studied in the literature can be expressed as special cases of the VCSP with restricted languages. We will discuss algorithms and complexity classifications for them in Section 5.

Example 10 (CSP)**.** The standard constraint satisfaction problem (CSP) over any fixed set of possible labels D can be seen as the special case of the VCSP where all cost functions take only the values 0 or ∞, representing allowed (satisfying) and disallowed tuples, respectively. Such constraints and cost functions are sometimes called *crisp*. In other words, the CSP can be seen as VCSP(Γ_{crisp}), where Γ_{crisp} is the language consisting of all cost functions on some fixed set D with range $\{0, \infty\}$. Note that the CSP can also be cast as the homomorphism problem for relational structures [29] (cf. Example 11).

Since the CSP includes many known NP-hard problems, such as NAE-SAT (Example 5) and Graph-3-Colouring, the language Γ_{crisp} is clearly intractable. However, many tractable subsets of Γ_{crisp} have been identified [77, 52, 29, 11, 7, 12, 49, 3, 4], mostly through an algebraic approach whose extension we discuss in Section 4. There are many surveys on the complexity of the CSP, see the books [25, 26], and also [14, 42].

Feder and Vardi conjectured that the CSP exhibits a *dichotomy*: that is, every finite language $\Gamma \subseteq \Gamma_{\mathsf{crisp}}$ is either tractable or intractable [29], thus excluding problems of intermediate complexity, as given by Ladner's Theorem (assuming P≠NP) [66]. The *Algebraic Dichotomy* conjecture, which we state formally and discuss in Section 5, specifies the precise boundary between tractable and intractable crisp languages [11].

Example 11 (Graph Homomorphism)**.** Given two digraphs $G = (V(G), E(G))$ and $H = (V(H), E(H))$, a mapping $f : V(G) \to V(H)$ is a *homomorphism* from G to H if f preserves edges, that is, $(u, v) \in E(G)$ implies $(f(u), f(v)) \in E(H)$.

The problem whether an input digraph G admits a homomorphism to a fixed digraph H is also known as the H-Colouring problem and has been actively studied in graph theory [41, 42].

For any graph H, let $D = V(H)$ and let Γ_H be the language that contains just the single binary cost function $\phi_H : D^2 \to \overline{\mathbb{Q}}$ defined by

$$\phi_H(x, y) \stackrel{\text{def}}{=} \begin{cases} 0 & \text{if } (x, y) \in E(H) \\ \infty & \text{otherwise} \end{cases} .$$

For any digraph H, the problem VCSP(Γ_H), which is a special case of the CSP (Example 10), corresponds to the H-colouring problem, where the input graph G is given by the scopes of the constraints. If we add all unary crisp functions to Γ_H then the resulting VCSP is known as List H-Colouring [41, 42].

It is known that both the Feder-Vardi conjecture and the Algebraic Dichotomy conjecture are equivalent to their restrictions to the H-colouring problem [13, 29].

Example 12 (Max-CSP)**.** An instance of the (weighted) maximum constraint satisfaction problem (Max-CSP) is an instance of the CSP where the goal is to maximise the (weighted) number of satisfied constraints.

When seeking the optimal solution, maximising the number of satisfied constraints is the same as minimising the number of unsatisfied constraints. Hence for any instance Φ of the Max-CSP, we can define a corresponding VCSP instance Φ' in which each constraint c of Φ is associated with a constraint over the same scope in Φ' which assigns cost 0 to tuples allowed by c, and cost 1 to tuples disallowed by c. It follows that Max-CSP is equivalent to VCSP(Γ_{Max}), where Γ_{Max} is the language consisting of cost functions whose values are restricted to zero and one.

For $D = \{0, 1\}$, the complexity of all subsets of Γ_{Max} has been completely classified in [58]. Initial results for languages over arbitrary finite sets appeared in [15]. A complete complexity classification will be discussed in Section 5.

Example 13 (Min-Cost-Hom)**.** Let Γ_{unary} consist of all unary cost functions and let $\Gamma_{\mathsf{mc}} = \Gamma_{\mathsf{crisp}} \cup \Gamma_{\mathsf{unary}}$ (where Γ_{crisp} is defined in Example 10). Problems of the form VCSP(Γ) with $\Gamma \subseteq \Gamma_{\mathsf{mc}}$ have been studied under the name of the Minimum-Cost Homomorphism problem (or Min-Cost-Hom) [39, 43, 81, 80, 85, 86]. Note that the first three of these papers assume that $\Gamma_{\mathsf{unary}} \subseteq \Gamma$, while the last three do not. In [39, 43] Γ is assumed to be of the form $\{\phi_H\} \cup \Gamma_{\mathsf{unary}}$, where ϕ_H is a binary crisp cost function, as in Example 11.

In any instance of VCSP(Γ_{mc}), the crisp constraints specify the CSP part, i.e., the feasibility aspect of the problem, while the unary constraints specify the optimisation aspect. More precisely, the unary constraints specify the costs of assigning labels to individual variables. Complexity classifications for special cases of Min-Cost-Hom will be discussed in Section 5.

Example 14 (Min-Ones)**.** An instance of the Boolean Minimum Ones (Min-Ones) problem is an instance of the CSP over $D = \{0, 1\}$ where the goal is to satisfy all constraints and minimise the number of variables assigned the label 1. Such instances correspond to Min-Cost-Hom instances over $\{0, 1\}$ in which all unary constraints are of the form η_0^1 as defined in Example 8 (which impose a unit cost for any variables assigned the label 1). A classification of the complexity of all subsets of this language was obtained in [25].

Example 15 (Min-Sol)**.** The Minimum Solution problem (Min-Sol) [53, 54] is a generalisation of Min-Ones from Example 14 to larger sets of labels where the only allowed unary cost function is a particular finite-valued injective function. Thus, this problem is also a subproblem of Min-Cost-Hom. Known complexity classifications for Min-Sol problems will be discussed in Section 5.

3 Polymorphisms and weighted polymorphisms

To develop general tools to classify the complexity of different valued constraint languages, we will now define certain algebraic properties of cost functions.

A function $f : D^k \to D$ is called a k-ary *operation* on D. The k-ary *projections*, defined for all $1 \leq i \leq k$, are the operations $\mathrm{e}_i^{(k)}$ such that $\mathrm{e}_i^{(k)}(x_1, \ldots, x_k) = x_i$. For any tuples $\mathbf{x_1}, \ldots, \mathbf{x_k} \in D^m$, we denote by $f(\mathbf{x_1}, \ldots, \mathbf{x_k})$ the tuple in D^m obtained by applying f to $\mathbf{x_1}, \ldots, \mathbf{x_k}$ componentwise.

Any valued constraint language Γ defined on D can be associated with a set of operations on D, known as the polymorphisms of Γ, and defined as follows.

Definition 16 (Polymorphism)**.** Let $\phi : D^m \to \overline{\mathbb{Q}}$ be a cost function and let $\text{Feas}(\phi) = \{\mathbf{x} \in D^m \mid \phi(\mathbf{x}) \text{ is finite}\}$ be the *feasibility relation* of ϕ. We say that an operation $f : D^k \to D$ is a *polymorphism* of ϕ if, for any $\mathbf{x_1}, \mathbf{x_2}, \ldots, \mathbf{x_k} \in \text{Feas}(\phi)$ we have that $f(\mathbf{x_1}, \mathbf{x_2}, \ldots, \mathbf{x_k}) \in \text{Feas}(\phi)$.

For any valued constraint language Γ over a set D, we denote by $\text{Pol}(\Gamma)$ the set of all operations on D which are polymorphisms of all $\phi \in \Gamma$. We denote by $\text{Pol}^{(k)}(\Gamma)$ the k-ary operations in $\text{Pol}(\Gamma)$.

Note that the projections are polymorphisms of all valued constraint languages.

For $\{0, \infty\}$-valued cost functions (relations) this notion of polymorphism corresponds precisely to the standard notion of polymorphism for relations [5, 52]. This notion of polymorphism has played a key role in the analysis of complexity for the CSP [52, 11]. However, for the analysis of the VCSP we need a more flexible notion that assigns weights to a collection of polymorphisms.

Definition 17 (Weighted Polymorphism)**.** Let $\phi : D^m \to \overline{\mathbb{Q}}$ be a cost function and let $C \subseteq \text{Pol}^{(k)}(\phi)$ be a collection of k-ary polymorphisms. A function $\omega : C \to \mathbb{Q}$ is called a k-ary *weighted polymorphism* of ϕ on C if it satisfies the following conditions:

- $\sum_{f \in C} \omega(f) = 0$;
- if $\omega(f) < 0$, then f is a projection;
- for any $\mathbf{x_1}, \mathbf{x_2}, \ldots, \mathbf{x_k} \in \text{Feas}(\phi)$

$$\sum_{f \in C} \omega(f)\phi(f(\mathbf{x_1}, \ldots, \mathbf{x_k})) \;\leq\; 0\,. \tag{2}$$

We define $\text{supp}(\omega) = \{f \mid \omega(f) > 0)$ to be the *positive support* of ω.

Remark. The definition of a weighted polymorphism can be re-stated in probabilistic terms, as follows. Consider Inequality (2) and assume that it is non-trivial, i.e., not all weights $\omega(f)$ are equal to 0. Let c be the smallest (negative) weight $\omega(f)$ that appears there. Add $\sum_{i=1}^{k} |c| \cdot \phi(\mathrm{e}_i^{(k)}(\mathbf{x_1}, \ldots, \mathbf{x_k})) = \sum_{i=1}^{k} |c| \cdot \phi(\mathbf{x}_i)$ to both sides of Inequality (2). Note that all weights of operations on the left-hand side are now non-negative. Normalise by dividing both sides by $|c| \cdot k$ and view the (new) weights of operations on the left-hand side as a probability distribution μ over a subset of $\text{Pol}^{(k)}(\phi)$. We can then re-write Inequality (2) as follows:

$$\mathbb{E}_{f \sim \mu}[\phi(f(\mathbf{x_1}, \ldots, \mathbf{x_k}))] \leq \text{avg}\{\phi(\mathbf{x_1}), \ldots, \phi(\mathbf{x_k})\}. \tag{3}$$

Thus, one can identify (non-trivial) k-ary weighted polymorphisms of ϕ with probability distributions μ over subsets of $\text{Pol}^{(k)}(\phi)$ satisfying Inequality (3) for all $\mathbf{x_1}, \ldots, \mathbf{x_k} \in \text{Feas}(\phi)$.

This is illustrated in Figure 1, which should be read from left to right. Let $C = \{f_1, \ldots, f_n\} \subseteq \mathrm{Pol}^{(k)}(\phi)$ and let μ be a probability distribution on C. Starting with the m-tuples $\mathbf{x_1}, \ldots, \mathbf{x_k}$, we first apply operations $f_1, \ldots, f_n$ to these tuples componentwise, thus obtaining the m-tuples $\mathbf{x'_1}, \ldots, \mathbf{x'_n}$. Inequality 3 amounts to comparing the average of the values of ϕ applied to the tuples $\mathbf{x_1}, \ldots, \mathbf{x_k}$, which corresponds to projections, with the weighted sum of the values of ϕ applied to the tuples $\mathbf{x'_1}, \ldots, \mathbf{x'_n}$, which is the expected value of $\phi(f(\mathbf{x_1}, \ldots, \mathbf{x_k}))$ as f is drawn from μ.

$$
\begin{array}{lllll}
\mathbf{x_1} & \mathbf{x_1}[1] & \mathbf{x_1}[2] & \ldots & \mathbf{x_1}[m] \\
\mathbf{x_2} & \mathbf{x_2}[1] & \mathbf{x_2}[2] & \ldots & \mathbf{x_2}[m] \\
\vdots & & & \vdots & \\
\mathbf{x_k} & \mathbf{x_k}[1] & \mathbf{x_k}[2] & \ldots & \mathbf{x_k}[m] \\
\hline
\mathbf{x'_1} = f_1(\mathbf{x_1}, \ldots, \mathbf{x_k}) & \mathbf{x'_1}[1] & \mathbf{x'_1}[2] & \ldots & \mathbf{x'_1}[m] \\
\mathbf{x'_2} = f_2(\mathbf{x_1}, \ldots, \mathbf{x_k}) & \mathbf{x'_2}[1] & \mathbf{x'_2}[2] & \ldots & \mathbf{x'_2}[m] \\
\vdots & & & \vdots & \\
\mathbf{x'_n} = f_n(\mathbf{x_1}, \ldots, \mathbf{x_k}) & \mathbf{x'_n}[1] & \mathbf{x'_n}[2] & \ldots & \mathbf{x'_n}[m]
\end{array}
\quad \xrightarrow{\phi} \quad
\begin{array}{l}
\phi(\mathbf{x_1}) \\
\phi(\mathbf{x_2}) \\
\vdots \\
\phi(\mathbf{x_k}) \\
\hline
\phi(\mathbf{x'_1}) \\
\phi(\mathbf{x'_2}) \\
\vdots \\
\phi(\mathbf{x'_n})
\end{array}
\quad \begin{array}{l}
\left.\vphantom{\begin{array}{l}a\\a\\a\\a\end{array}}\right\} \frac{1}{k} \sum_{i=1}^{k} \phi(\mathbf{x_i}) \\
\\
\left.\vphantom{\begin{array}{l}a\\a\\a\\a\end{array}}\right\} \sum_{i=1}^{n} \Pr_{\mu}[f_i] \phi(\mathbf{x'_i})
\end{array}
\quad \text{IV}
$$

Figure 1: Probabilistic definition of a weighted polymorphism.

If ω is a weighted polymorphism of ϕ, then we say that ϕ *admits* ω as a weighted polymorphism. We say that a language Γ admits a weighted polymorphism ω if ω is a weighted polymorphism of every cost function $\phi \in \Gamma$.

Weighted polymorphisms were introduced in [19] and have allowed a general algebraic theory of complexity for valued constraints to be developed, as we will describe in Section 4.

Certain special kinds of weighted polymorphisms were introduced in earlier papers, but have now been subsumed by the more general theory described here. For example, the notion of a *fractional polymorphism* was introduced in [16]. For finite-valued functions, this notion coincides with the notion of a weighted polymorphism.

A more restricted form of weighted polymorphism was introduced earlier in [17] and is known as a *multimorphism*. This is essentially a k-ary weighted polymorphism where the values of $\omega(f)$ are all integers, and the values of $\omega(f)$ for projection operations are all equal to -1. Using the probabilistic view, this means that the probability of each operation in a k-ary weighted polymorphism is of the form ℓ/k where $\ell \in \mathbb{Z}$.

One can specify a k-ary multimorphism as a k-tuple $\mathbf{f} = \langle f_1, \ldots, f_k \rangle$ of k-ary operations f_i on D, where each operation f for which $\omega(f)$ is positive appears $\omega(f)$ times, and then the definition simplifies as follows: for all $\mathbf{x_1}, \ldots, \mathbf{x_k} \in D^m$,

$$\sum_{i=1}^{k} \phi(f_i(\mathbf{x_1}, \ldots, \mathbf{x_k})) \;\leq\; \sum_{i=1}^{k} \phi(\mathbf{x_i})\,. \tag{4}$$

Weighted polymorphisms (including the special cases of fractional polymorphisms and multimorphisms) have proved to be a valuable tool for identifying tractable valued constraint languages, as we will illustrate in this Section.

Example 18 (Submodularity)**.** For any finite set V, a rational-valued function h defined on subsets of V is called a *set function*. A set function h is called *submodular* if for all subsets S and T of V,

$$h(S \cap T) + h(S \cup T) \;\leq\; h(S) + h(T). \tag{5}$$

Submodular functions are a key concept in operational research and combinatorial optimisation (see, e.g. [30, 78, 84] for extensive information about them). They are often considered to be a discrete analogue of convex functions. Examples of submodular functions include cuts in graphs, matroid rank functions, and entropy functions. There are combinatorial algorithms for minimising submodular functions in polynomial time (see [78, 30], and also [51]).

If we set $D = \{0, 1\}$, then any set function h on V can be associated with a ($|V|$-ary) cost function ϕ defined on the characteristic vectors of subsets of V. The union and intersection operations on subsets correspond to the Min and Max operations on the associated characteristic vectors. Hence h is submodular if and only if the associated cost function ϕ satisfies the following inequality:

$$\phi(\mathrm{Min}(\mathbf{x}_1, \mathbf{x}_2)) + \phi(\mathrm{Max}(\mathbf{x}_1, \mathbf{x}_2)) - \phi(\mathbf{x}_1) - \phi(\mathbf{x}_2) \leq 0\,.$$

But this means that ϕ admits the 2-ary weighted polymorphism ω_{sub}, defined by:

$$\omega_{sub}(f) \;\stackrel{\text{def}}{=}\; \begin{cases} -1 & \text{if } f \in \{e_1^{(2)}, e_2^{(2)}\} \\ +1 & \text{if } f \in \{\mathrm{Min}, \mathrm{Max}\} \\ \;\;0 & \text{otherwise.} \end{cases}\,.$$

This is equivalent to saying that ϕ admits $\langle \mathrm{Min}, \mathrm{Max} \rangle$ as a multimorphism.

Example 19 (Generalised Submodularity)**.** Let D be a finite *lattice*, i.e., a partially ordered set, where each pair of elements $\{a, b\}$ has a least upper bound, $\vee(a, b)$, and a greatest lower bound, $\wedge(a, b)$. We denote by Γ_{sub} the set of all cost functions over D that admit $\langle \vee, \wedge \rangle$ as a multimorphism. Using a polynomial-time strongly combinatorial algorithm for minimising submodular functions, it was shown in [17] that Γ_{sub} is tractable when D is a totally ordered lattice (i.e., a *chain*). More general lattices will be discussed in Section 5 and Section 7.

Example 20 (Max)**.** We denote by Γ_{max} the set of all cost functions (over some fixed finite totally ordered set D) that admit $\langle \mathrm{Max}, \mathrm{Max} \rangle$ as a multimorphism, where $\mathrm{Max} : D^2 \to D$ is the binary operation returning the larger of its two arguments. Note that Γ_{max} includes all monotonic decreasing finite-valued cost functions, as well as some non-monotonic crisp cost functions [17]. It was shown in [17] that Γ_{max} is tractable.

Example 21 (Min)**.** We denote by Γ_{min} the set of all cost functions (over some fixed finite totally ordered set D) that admit $\langle \mathrm{Min}, \mathrm{Min} \rangle$ as a multimorphism, where $\mathrm{Min} : D^2 \to D$ is the binary operation returning the smaller of its two arguments. The tractability of Γ_{min} was established in [17].

Example 22 (Bisubmodularity)**.** For a given finite set V, bisubmodular functions are functions defined on pairs of disjoint subsets of V with a requirement similar to Inequality 5 (see [30, 71] for the precise definition). Examples of bisubmodular functions include rank functions of delta-matroids [30].

A property equivalent to bisubmodularity can be defined on cost functions on the set $D = \{0, 1, 2\}$. We define two binary operations Min_0 and Max_0 as follows:

$$\mathrm{Min}_0(x, y) \stackrel{\text{def}}{=} \begin{cases} 0 & \text{if } 0 \neq x \neq y \neq 0 \\ \mathrm{Min}(x, y) & \text{otherwise} \end{cases},$$

$$\mathrm{Max}_0(x, y) \stackrel{\text{def}}{=} \begin{cases} 0 & \text{if } 0 \neq x \neq y \neq 0 \\ \mathrm{Max}(x, y) & \text{otherwise} \end{cases}.$$

We denote by Γ_{bis} the set of finite-valued cost functions that admit $\langle \mathrm{Min}_0, \mathrm{Max}_0 \rangle$ as a multimorphism. The language Γ_{bis} can be shown to be tractable using the results of [71] (see also [30]).

The definitions of Min_0 and Max_0 still make sense when $D = \{0, 1, 2 \ldots, k\}$, $k \geq 3$. In that case, functions on D that admit $\langle \mathrm{Min}_0, \mathrm{Max}_0 \rangle$ as a multimorphism are called *k-submodular*; they were introduced in [46].

Example 23 (Skew Bisubmodularity)**.** Let $D = \{0, 1, 2\}$. Recall the definition of operations Min_0 and Max_0 from Example 22. We define

$$\mathrm{Max}_1(x, y) \stackrel{\text{def}}{=} \begin{cases} 1 & \text{if } 0 \neq x \neq y \neq 0 \\ \mathrm{Max}(x, y) & \text{otherwise} \end{cases}.$$

A function ϕ: $D^m \to \overline{\mathbb{Q}}$ is called *α-bisubmodular* [48], for some real $0 < \alpha \leq 1$, if ϕ admits the weighted polymorphism ω defined by $\omega(\mathrm{Min}_0) = 1$, $\omega(\mathrm{Max}_0) = \alpha$, $\omega(\mathrm{Max}_1) = (1 - \alpha)$, and $\omega(e_1^{(2)}) = \omega(e_2^{(2)}) = -1$. Note that 1-bisubmodular functions are (ordinary) bisubmodular functions as defined in Example 22. It is shown in [48] that each distinct value of α is associated with a distinct class of α-bisubmodular functions. The tractability of α-bisubmodular valued constraint languages will be discussed in Section 5.

Example 24 ((Symmetric) Tournament Pair)**.** A binary operation $f : D^2 \to D$ is called a *tournament* operation if (i) f is commutative, i.e., $f(x, y) = f(y, x)$ for all $x, y \in D$; and (ii) f is conservative, i.e., $f(x, y) \in \{x, y\}$ for all $x, y \in D$. The *dual* of a tournament operation is the unique tournament operation g satisfying $x \neq y \Rightarrow g(x, y) \neq f(x, y)$.

A *tournament pair* is a pair $\langle f, g\rangle$, where both f and g are tournament operations. A tournament pair $\langle f, g\rangle$ is called *symmetric* if g is the dual of f.

Let Γ be an arbitrary language that admits a symmetric tournament pair as a multimorphism. It was shown in [18], by a reduction to the minimisation problem for submodular functions (cf. Example 19), that any such Γ is tractable. It is shown in [62] that any finite-valued language that admits a symmetric tournament pair multimorphism also admits the submodularity multimorphism with respect to some totally ordered lattice on D (cf. Example 19).

Now let Γ be an arbitrary language that admits any tournament pair as a multimorphism. It was shown in [18], by a reduction to the symmetric tournament pair case, that any such Γ is also tractable.

Example 25 (1-Defect)**.** Let b and c be two distinct elements of D and let $(D; <)$ be a partial order which relates all pairs of elements except for b and c. We call $\langle f, g\rangle$, where $f, g : D^2 \to D$ are two binary operations, a *1-defect* if f and g are both commutative and satisfy the following conditions:

- If $\{x, y\} \neq \{b, c\}$, then $f(x, y) = \mathrm{Min}(x, y)$ and $g(x, y) = \mathrm{Max}(x, y)$.
- If $\{x, y\} = \{b, c\}$, then $\{f(x, y), g(x, y)\} \cap \{x, y\} = \emptyset$, and $f(x, y) < g(x, y)$.

The tractability of languages that admit a 1-defect multimorphism was shown in [57], and was used in the classification of the Max-CSP over a four-element set (see Section 5).

Example 26 (Majority)**.** A ternary operation $f : D^3 \to D$ is called a majority operation if $f(x, x, y) = f(x, y, x) = f(y, x, x) = x$ for all $x, y \in D$.

Let $\mathbf{f} = \langle f_1, f_2, f_3\rangle$ be a triple of ternary operations such that f_1, f_2 and f_3 are all majority operations. Let $\phi : D^m \to \overline{\mathbb{Q}}$ be an m-ary cost function that admits $\mathbf{f}$ as a multimorphism. By Inequality (4), for all $\mathbf{x}, \mathbf{y} \in D^m$, $3\phi(\mathbf{x}) \leq \phi(\mathbf{x}) + \phi(\mathbf{x}) + \phi(\mathbf{y})$ and $3\phi(\mathbf{y}) \leq \phi(\mathbf{y}) + \phi(\mathbf{y}) + \phi(\mathbf{x})$. Therefore, if both $\phi(\mathbf{x})$ and $\phi(\mathbf{y})$ are finite, then we have $\phi(\mathbf{x}) \leq \phi(\mathbf{y})$ and $\phi(\mathbf{y}) \leq \phi(\mathbf{x})$, and hence $\phi(\mathbf{x}) = \phi(\mathbf{y})$. In other words, the range of ϕ is $\{c, \infty\}$, for some finite $c \in \overline{\mathbb{Q}}$.

Let Γ_{Mjty} be the set of all cost functions that admit as a multimorphism some triple $\mathbf{f} = \langle f_1, f_2, f_3\rangle$ of arbitrary ternary majority operations. The tractability of Γ_{Mjty} was shown in [17].

Example 27 (Minority)**.** A ternary operation $f : D^3 \to D$ is called a minority operation if $f(x,x,y) = f(x,y,x) = f(y,x,x) = y$ for all $x, y \in D$. Let Γ_{Mnty} be the set of cost functions that admit as a multimorphism some triple $\mathbf{f} = \langle f_1, f_2, f_3 \rangle$ of arbitrary ternary minority operations. A similar argument to the one in Example 26 shows that the cost functions in Γ_{Mnty} have range $\{c, \infty\}$, for some finite $c \in \overline{\mathbb{Q}}$. The tractability of Γ_{Mnty} was shown in [17].

Example 28 (MJN)**.** Let $\mathbf{f} = \langle f_1, f_2, f_3 \rangle$ be three ternary operations such that f_1 and f_2 are majority operations, and f_3 is a minority operation. Let Γ_{MJN} be the set of cost functions that admit $\mathbf{f}$ as a multimorphism. The tractability of Γ_{MJN} was shown in [63], generalising an earlier tractability result for a specific $\mathbf{f}$ of this form from [17].

Other tractable valued constraint languages defined by weighted polymorphisms include the so-called $L^\#$-convex languages [30], as well as the weakly and strongly tree-submodular languages defined in [60]. Hirai [45] recently introduced a framework of submodular functions on modular semilattices (defined by a type of weighted polymorphism) that generalises many examples given above, including standard submodularity, k-submodularity, skew bisubmodularity, and tree submodularity. See [45] for the natural, but somewhat technical, definition of this very general framework.

4 A general algebraic theory of complexity

We have seen in the previous section that many tractable cases of the VCSP can be defined by having a particular weighted polymorphism. The algebraic theory developed in [19] establishes that, in fact, every tractable valued constraint language can be exactly characterised by its weighted polymorphisms. This extends (parts of) the algebraic theory previously developed for the CSP [10, 11, 52] that has led to significant advances in understanding the landscape of complexity for the CSP over the last 10 years (e.g., [2, 3, 4, 7, 8, 9, 12, 49, 67]). In this section, we will give a brief overview of the main results of this new algebraic theory for the VCSP. We refer the reader to [19] for full details and proofs.

We first recall some basic terminology from universal algebra [5, 79]. We denote by $\mathbf{O}_D$ the set of all finitary operations on D and by $\mathbf{O}_D^{(k)}$ the k-ary operations in $\mathbf{O}_D$. Let $f \in \mathbf{O}_D^{(k)}$ and $g_1, \ldots, g_k \in \mathbf{O}_D^{(\ell)}$. The *superposition* of f and $g_1, \ldots, g_k$ is the ℓ-ary operation $f[g_1, \ldots, g_k]$ such that $f[g_1, \ldots, g_k](x_1, \ldots, x_\ell) = f(g_1(x_1, \ldots, x_\ell), \ldots, g_k(x_1 \ldots, x_\ell))$.

A set $F \subseteq \mathbf{O}_D$ is called a *clone* of operations if it contains all the projections on D and is closed under superposition. It is easy to verify that the set of operations $\mathrm{Pol}(\Gamma)$ is a clone. Clones are actively studied in universal algebra; for example,

all (countably many) clones on $D = \{0, 1\}$ are known, but the situation is known to be much more complicated for larger sets D (see, e.g., [5, 79]).

For each $F \subseteq \mathbf{O}_D$ we define $\mathrm{Clone}(F)$ to be the smallest clone containing F. For any clone C, we use $C^{(k)}$ to denote the k-ary operations in C.

Now we consider the effect of extending a valued constraint language $\Gamma \subseteq \mathbf{\Phi}_D$ to a possibly larger valued constraint language. We first define and study a notion of *expressibility* for valued constraint languages. This notion has played a key role in the analysis of complexity for the CSP and VCSP [11, 52, 17, 89].

Definition 29. We say that an m-ary cost function ϕ is *expressible* over a constraint language Γ if there exists a instance $\Phi \in \mathrm{VCSP}(\Gamma)$ with variables $V = \{x_1, \ldots, x_n, y_1, \ldots, y_m\}$, such that

$$\phi(y_1, \ldots, y_m) = \min_{x_1, \ldots, x_n} \Phi(x_1, \ldots, x_n, y_1, \ldots, y_m)\,.$$

Definition 30. A valued constraint language $\Gamma \subseteq \mathbf{\Phi}_D$ is called a *weighted relational clone* if it is closed under expressibility, scaling by non-negative rational constants, and addition of rational constants. We define $\mathrm{wRelClone}(\Gamma)$ to be the smallest weighted relational clone containing Γ.

Theorem 31 ([19])**.** *A valued constraint language* Γ *is tractable if* $\mathrm{wRelClone}(\Gamma)$ *is tractable and intractable if* $\mathrm{wRelClone}(\Gamma)$ *is intractable.*

Example 32. By Theorem 31, and Examples 5 and 6, in order to show that Γ is an intractable language it is sufficient to show that ϕ_{nae} or ϕ_{xor} is in $\mathrm{wRelClone}(\Gamma)$. We discuss general reasons for intractability of constraint languages in Section 5.

We now develop tools that will allow an alternative characterisation of any weighted relational clone.

Definition 33. We define a k-ary *weighting* of a clone C to be a function $\omega : C^{(k)} \to \mathbb{Q}$ such that $\omega(f) < 0$ only if f is a projection and

$$\sum_{f \in C^{(k)}} \omega(f) = 0\,.$$

We denote by $\mathbf{W}_C$ the set of all possible weightings of C and by $\mathbf{W}_C^{(k)}$ the set of k-ary weightings of C.

Since a weighting is simply a rational-valued function satisfying certain linear inequalities it can be scaled by any non-negative rational to obtain a new weighting. Similarly, any two weightings of the same clone of the same arity can be added to obtain a new weighting of that clone.

The notion of superposition can also be extended to weightings in a natural way, by forming a superposition with each argument of the weighting, as follows.

Definition 34. For any clone C, any $\omega \in \mathbf{W}_C^{(k)}$ and any $g_1, g_2, \ldots, g_k \in C^{(\ell)}$, we define the *superposition* of ω and $g_1, \ldots, g_k$, to be the function $\omega[g_1, \ldots, g_k] : C^{(\ell)} \to \mathbb{Q}$ defined by

$$\omega[g_1, \ldots, g_k](f') \stackrel{\text{def}}{=} \sum_{\substack{f \in C^{(k)} \\ f[g_1,\ldots,g_k]=f'}} \omega(f)\,. \tag{6}$$

It follows immediately from the definition of superposition that the sum of the weights in any superposition $\omega[g_1, \ldots, g_k]$ is equal to the sum of the weights in ω, which is zero, by Definition 33. However, it is not always the case that an arbitrary superposition satisfies the other condition in Definition 33, that negative weights are only assigned to projections. Hence we make the following definition:

Definition 35. If the result of a superposition is a valid weighting, then that superposition will be called a *proper* superposition.

Definition 36. A *weighted clone*, W, is a non-empty set of weightings of some fixed clone C which is closed under non-negative scaling, addition of weightings of equal arity, and proper superposition with operations from C. The clone C is called the *support* of W.

Example 37. For any clone, C, the set $\mathbf{W}_C$ containing all possible weightings of C is a weighted clone with support C.

Example 38. For any clone, C, the set $\mathbf{W}_C^0$ containing all *zero-valued* weightings of C is a weighted clone with support C. $\mathbf{W}_C^0$ contains exactly one weighting of each possible arity, which assigns the value 0 to all operations in C of that arity.

Weighted clones were introduced only very recently and not much is known about them (in comparison with ordinary clones). Some initial study of weighted clones can be found in [19, 24].

Given a cost function ϕ, some weightings will satisfy the conditions of Definition 17, and hence be weighted polymorphisms of ϕ.

Definition 39. For any $\Gamma \subseteq \mathbf{\Phi}_D$, we denote by $\mathrm{wPol}(\Gamma)$ the set of all weightings of $\mathrm{Pol}(\Gamma)$ which are weighted polymorphisms of all cost functions $\phi \in \Gamma$.

To define a mapping in the other direction, we need to consider the union of the sets $\mathbf{W}_C$ over all clones C on some fixed set D, which will be denoted $\mathbf{W}_D$. If we have a set $W \subseteq \mathbf{W}_D$ which may contain weightings of *different* clones over D, then we can extend each of these weightings with zeros, as necessary, so that they are weightings of the same clone C, where C is the smallest clone containing all the clones that are supports of weightings in W. For any set $W \subseteq \mathbf{W}_D$, we define $\mathrm{wClone}(W)$ to be the smallest weighted clone containing this set of extended weightings obtained from W.

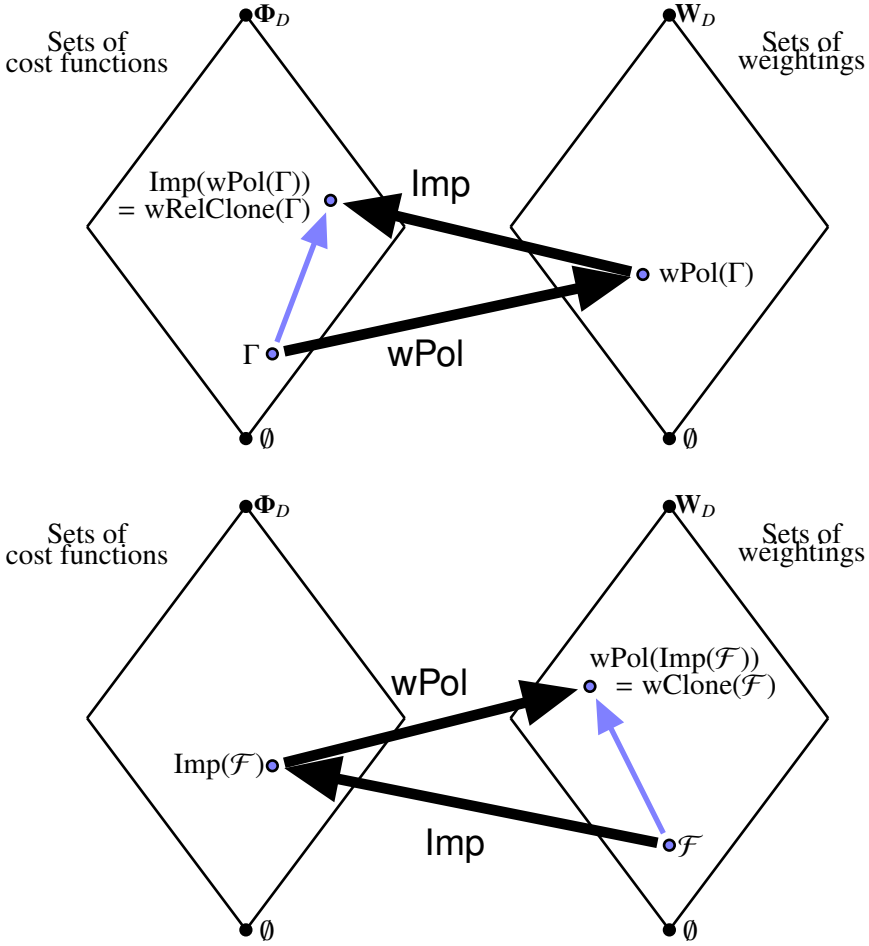

Figure 2: Galois connection between $\mathbf{\Phi}_D$ and $\mathbf{W}_D$.

Definition 40. For any $W \subseteq \mathbf{W}_D$, we denote by $\mathrm{Imp}(W)$ the set of all cost functions in $\mathbf{\Phi}_D$ which admit all weightings $\omega \in W$ as weighted polymorphisms [1].

It follows immediately from the definition of a Galois connection [5] that, for any set D, the mappings wPol and Imp form a Galois connection between $\mathbf{W}_D$ and $\mathbf{\Phi}_D$, as illustrated in Figure 2. A characterisation of this Galois connection for finite sets D is given by the following theorem from [19]:

Theorem 41 (Galois Connection for Valued Constraint Languages [19])**.**

1. *For any finite D, and any finite $\Gamma \subseteq \mathbf{\Phi}_D$,* $\mathrm{Imp}(\mathrm{wPol}(\Gamma)) = \mathrm{wRelClone}(\Gamma)$.
2. *For any finite D and any finite $W \subseteq \mathbf{W}_D$,* $\mathrm{wPol}(\mathrm{Imp}(W)) = \mathrm{wClone}(W)$.

[1] The name Imp is chosen to suggest that such cost functions are *improved* by weightings in W.

It follows that to identify all tractable valued constraint languages on a finite set D it is sufficient to study the possible weighted clones on D. This provides a new approach to the identification of tractable cases, which we hope will prove to be as successful as the algebraic approach has been in the study of the CSP.

The Galois connection described in Theorem 41 can be used to derive necessary conditions for tractability. It is shown in [19] that every tractable valued constraint language must have a weighted polymorphism that assigns positive weight to certain specific kinds of operations.

The algebraic theory of the CSP extends beyond clones to finite algebras and varieties of algebras (see [10, 11, 67], see also the surveys in [26]). This extension explains why the complexity of a (crisp) language is determined by the identities satisfied by its polymorphisms, which is why we usually define the relevant operations by identities. This extension was instrumental in obtaining most state-of-the-art results in this area (e.g. [2, 3, 4, 7, 8, 9, 12, 49, 67]). An initial study of a similar extension of the algebraic theory for the VCSP can be found in [73].

A valued constraint language Γ is called a *core* if every unary weighted polymorphism ω of Γ has the property that every operation $f \in \mathrm{supp}(\omega)$ is surjective. Intuitively, a valued constraint language Γ defined on D is a core if no label $x \in D$ can be removed without losing solutions. In other words, for every $a \in D$ there is an instance $\Phi_a \in \mathrm{VCSP}(\Gamma)$ such that a appears in every optimal solution to Φ_a [48]. Furthermore, a language Γ is called a *rigid core* if $\mathrm{Pol}^{(1)}(\Gamma)$ contains only the unary projection $e_1^{(1)}$. In this case, all operations in $\mathrm{Pol}(\Gamma)$ must be *idempotent*, i.e., satisfy the identity $f(x, \ldots, x) = x$.

Generalising the arguments used for the CSP [11] and finite-valued languages [48, 83], one can show that the search for tractable valued constraint languages can be restricted to languages that are rigid cores, see [73]. This technical restriction has very important implications because the structural theory of finite algebras works much better for idempotent operations (and idempotent algebras and varieties), see, e.g. [2, 3, 4, 7, 8, 12, 49, 67, 69]

5 Algorithms and complexity classifications

A curious feature of research into the tractability of constraint languages is that all languages known to be tractable have been shown tractable by using very few algorithmic techniques.

Despite many tractability results concerning crisp languages (i.e., the CSP), only two algorithmic techniques seem to be sufficient, and the applicability of each of them individually has been characterised by specific algebraic conditions.

The first technique is based on enforcing local consistency, which is a natural algorithm for dealing with (crisp) constraints. Roughly, this algorithm, for a given CSP instance, starts by adding a new constraint for each subset of variables of bounded size, the new constraints initially allowing all tuples. Then the algorithm repeatedly discards (i.e., disallows) tuples of labels in the new constraints that are inconsistent with at least one constraint in the instance. Eventually, either all assignments are discarded or else local consistency is established; this procedure takes polynomial time for any fixed D and any fixed bound on the size of subsets. The former case implies no feasible assignments. One says that a CSP is solved by local consistency if the latter case implies the existence of a feasible assignment. The power of local consistency (i.e., a precise characterisation of crisp languages that give rise to VCSP instances solvable by some form of local consistency) has recently been established [4, 8]. A k-ary ($k \geq 2$) idempotent operation $f : D^k \to D$ is called a *weak near-unanimity* operation if, for all $x, y \in D$,

$$f(y, x, x, \ldots, x) = f(x, y, x, x, \ldots, x) = f(x, x, \ldots, x, y).$$

Theorem 42 (Bounded Width [4, 8])**.** *Let* Γ *be a crisp language that is a rigid core.* VCSP(Γ) *is solvable by local consistency if and only if* Pol(Γ) *contains weak near-unanimity operations of all but finitely many arities.*

Remark. One of many equivalent forms of the Algebraic Dichotomy conjecture [11] mentioned in Example 10 is the following: A crisp language Γ that is a rigid core is tractable if and only if Pol(Γ) contains a weak near-unanimity operation. Crisp rigid cores Γ that do not satisfy this condition are known to be NP-complete [11]. This reformulation of the conjecture follows from [69] via [10] (see also [3]).

The second standard algorithmic technique for the CSP is based on the property of having a polynomial-sized representation (a generating set) for the solution set of any instance [9, 49]. Roughly, the algorithm works by starting from the empty set and adding constraints in an instance one by one while maintaining (in polynomial time) a small enough representation of the current solution set (of feasible assignments). At the end (i.e., after all constraints have been added), either this representation is non-empty and contains a solution to the instance or else there is no solution. In a way, this technique is a generalisation of Gaussian elimination. This algorithm is often called "few subpowers" because it is related to a certain algebraic property to do with the number of of subalgebras in powers of an algebra. The power of this algorithm was established in [49]. A k-ary ($k \geq 3$) operation $f : D^k \to D$ is called an *edge* operation if, for all $x, y \in D$,

$$f(y, y, x, x, \ldots, x) = f(y, x, y, x, x, \ldots, x) = x$$

and

$$f(x, x, x, y, x, \ldots, x) = f(x, x, x, x, y, x, \ldots, x) = f(x, \ldots, x, y) = x.$$

Theorem 43 (Few Subpowers [49])**.** *Let* Γ *be a crisp language. Then* VCSP(Γ) *is solvable by the few subpowers algorithm if* Pol(Γ) *contains an edge operation.*

The converse to this theorem is true in the following sense: the absence of edge operations from Pol(Γ) implies that the presence of small enough representations is not guaranteed, see [49] for details. Interestingly, the few subpowers algorithm makes use of the actual edge operations in its work (in contrast with bounded width, where the weak near-unanimity operations only guarantee correctness).

It is natural to try to extend the conditions characterising the applicability of these two algorithms to the VCSP, and to investigate whether valued constraint languages satisfying these algebraic conditions are also tractable. However, so far this approach is largely unexplored. Some forms of local consistency techniques have been generalised to the VCSP [20], but their power is not fully understood.

For the general VCSP another algorithm, based on linear programming, has been the most thoroughly investigated. Every VCSP instance has a natural linear programming relaxation called the *basic LP relaxation* (BLP). For an instance Φ defined by $\Phi(\mathbf{x}) = \sum_{i=1}^{q} \phi_i(\mathbf{x}_i)$, with set of variables V, the associated LP instance BLP(Φ) is defined as follows:

$$
\begin{aligned}
BLP(\Phi) \stackrel{\text{def}}{=} \min \quad & \sum_{i=1}^{q} \sum_{\mathbf{s_i} \in D^{\mathbf{x}_i}} \phi_i(\mathbf{s_i})\, \lambda_{i,\mathbf{s_i}} && \text{(7a)}\\
\text{s.t.} \quad & \sum_{\mathbf{s_i} \in D^{\mathbf{x_i}} \mid \mathbf{s_i}(x)=a} \lambda_{i,\mathbf{s_i}} = \mu_x(a), \quad 1 \le i \le q,\ x \in \mathbf{x_i},\ a \in D && \text{(7b)}\\
& \sum_{a \in D} \mu_x(a) = 1, \quad x \in V && \text{(7c)}\\
& \lambda_{i,\mathbf{s_i}} = 0, \quad 1 \le i \le q,\ \phi_i(\mathbf{s_i}) = \infty && \text{(7d)}
\end{aligned}
$$

We minimise over the variables $\mu_x(a)$, where $x \in V$ and $a \in D$, and $\lambda_{i,\mathbf{s_i}}$, where $1 \le i \le q$ and $\mathbf{s_i} \in D^{\mathbf{x}_i}$, that take on real values in the interval $[0, 1]$. These variables can be seen as probability distributions on D and $D^{\mathbf{x}_i}$, respectively. The marginalization constraints (7b) impose that μ_x is the marginal of $\lambda_{i,\mathbf{s_i}}$, for each constraint and each variable x in the scope of that constraint. Note that terms in (7a) corresponding to (7d) are assumed to be equal to 0.

We remark that an LP relaxation of the VCSP, similar or closely related to (7), has been proposed independently by many authors; we refer the reader to [62] and the references therein.

Given a VCSP instance Φ, we say that BLP *solves* Φ if the optimal value of BLP(Φ) is equal to the optimal value of Φ. Moreover, we say that BLP solves a valued constraint language Γ if BLP solves every instance $\Phi \in$ VCSP(Γ). It is shown in [62] that in all cases where BLP solves Γ, a standard self-reduction

technique can be used to obtain an assignment that minimises any Φ in VCSP(Γ) in polynomial time. Hence if BLP solves Γ, then Γ is tractable.

The power of BLP for valued constraint languages was fully characterised in [82]. To state this result, we first introduce some further terminology about operations. A k-ary operation $f : D^k \to D$ is called *symmetric* if for every permutation π on $\{1, \ldots, k\}$, $f(x_1, \ldots, x_k) = f(x_{\pi(1)}, \ldots, x_{\pi(k)})$. A weighted polymorphism ω is called symmetric if supp(ω) is non-empty and contains symmetric operations only. Finally, we say that an operation f is *generated* from a set of operations $F \subseteq \mathbf{O}_D$ if $f \in \text{Clone}(F)$.

Theorem 44 (Power of BLP for Arbitrary Languages [82])**.** *Let Γ be a valued constraint language. Then the following are equivalent:*

1. *BLP solves Γ;*
2. *For every $k \geq 2$, Γ admits a k-ary symmetric weighted polymorphism;*
3. *For every $k \geq 2$, Γ admits a weighted polymorphism (not necessarily k-ary) ω_k such that $supp(\omega_k)$ generates a symmetric k-ary operation.*

It is unknown whether the conditions in Theorem 44 are decidable. Nevertheless, condition (3) has turned out to be very useful for proving the tractability of many valued constraint languages. A binary operation $f : D^2 \to D$ is called a *semilattice* operation if f is associative, commutative, and idempotent. Since any semilattice operation trivially generates symmetric operations of all arities, Theorem 44 shows that any valued constraint language with a binary weighted polymorphism whose positive support includes a semilattice operation is solvable using the BLP. This immediately implies that all of the following cases are solvable using the BLP, and hence tractable: languages with a (generalised) submodular multimorphism (Example 19), a bisubmodular multimorphism (Example 22), a symmetric tournament pair multimorphism (Example 24), or a skew bisubmodular weighted polymorphism (Example 23), or the weighted polymorphisms describing submodularity on modular semilattices [45]. Moreover, a not very difficult argument can be used to show that languages with a 1-defect multimorphism (Example 25) also satisfy condition (3) of Theorem 44 [82], and thus are tractable.

For valued constraint languages where the cost functions take only finite values, this result has been strengthened even further [82, 61], see also [62].

Theorem 45 (Power of BLP for Finite-Valued Languages [82, 61])**.** *Let Γ be a valued constraint language where every cost function takes only finite values. Then the following are equivalent:*

1. *BLP solves Γ;*

2. *For every $k \geq 2$, Γ admits a k-ary symmetric weighted polymorphism;*

3. *For some $k \geq 2$, Γ admits a k-ary symmetric weighted polymorphism;*

4. *Γ admits a binary symmetric weighted polymorphism;*

5. *Γ admits a weighted polymorphism ω such that* $\mathrm{supp}(\omega)$ *generates a symmetric operation.*

We mentioned above that the tractability of constraint languages seems to come from very few techniques. Interestingly, the hardness of constraint languages also seems to come from very few specific hard problems! Recall the functions ϕ_{nae} and ϕ_{xor} on $\{0,1\}$, from Examples 5 and 6, corresponding to the NP-hard problems NAE-SAT and Max-Cut.

The hardness of VCSP($\{\phi_{\mathsf{nae}}\}$) generalises in an obvious way to any problem VCSP($\{\phi\}$) over any set D, where ϕ is defined as follows: choose a subset $X \subseteq D$ with $|X| > 1$ and a surjective function $h : X \to \{0,1\}$, and let $\phi(x,y,z) = \phi_{\mathsf{nae}}(h(x),h(y),h(z))$ if $(x,y,z) \in X^3$ and $\phi(x,y,z) = \infty$ otherwise. Call such functions NAE-like. By Theorem 31, every language Γ such that wRelClone (Γ) contains a NAE-like function is intractable. Moreover, *every* crisp core language Γ known to be NP-complete satisfies this condition [11]. In other words, the ability to express ϕ_{nae} is the only known reason for a crisp core language to be NP-hard, and the only reason for this if the Algebraic Dichotomy conjecture holds.

Now let ϕ be a binary cost function over D such that, for some distinct $a, b \in D$, $\mathrm{argmin}(\phi) = \{(a,b),(b,a)\}$ and $\phi(a,a), \phi(b,b)$ are both finite. The hardness of VCSP($\{\phi_{\mathsf{xor}}\}$) on $\{0,1\}$ generalises in an obvious way to VCSP($\{\phi\}$) for such functions ϕ (see [48, 83]). Call such a function XOR-like. By Theorem 31, every Γ such that wRelClone (Γ) contains a XOR-like function is intractable. Moreover, the converse is known to be true, that is, for *every* NP-hard finite-valued core language Γ, wRelClone (Γ) contains a XOR-like function [48, 83] (see Theorem 46).

In fact, *most* languages (not necessarily crisp or finite-valued) known to be NP-hard are known to satisfy the condition that wRelClone (Γ) contains a function that is NAE-like or XOR-like. It is an open question whether there exist intractable languages Γ that do not satisfy this condition. Some NP-hard languages, e.g. those from [81], are not known to satisfy it.

We now focus on complexity classifications. For crisp languages (i.e. pure feasibility problems), complexity classifications have been established for languages over two-element sets [77] and three-element sets [7] and for languages containing all unary relations [12, 2]. For finite-valued languages (i.e. pure optimisation

problems), it has been shown that BLP solves *all* tractable cases [83].

Theorem 46 (Classification of Finite-Valued Languages [83])**.** *Let* Γ *be a finite-valued constraint language that is a core. Either* Γ *has a binary symmetric weighted polymorphism (and hence is solvable by BLP), or else* wRelClone(Γ) *contains a XOR-like function, and hence* Γ *is intractable.*

Theorem 46 generalises several previous classification results for finite-valued languages. Tractability in these earlier results was often characterised by (more) specific binary symmetric weighted polymorphisms:

- A core $\{0, 1\}$-valued language[2] over a two-element set [58, 25], or over a three-element set [55], or including all unary $\{0, 1\}$-valued functions [28] is tractable if it is submodular on a chain (cf. Examples 18 and 19), and intractable otherwise.
- A core $\{0, 1\}$-valued language over a four-element set [57] is tractable if it is submodular on some lattice (cf. Example 19) or 1-defect (cf. Example 25) and intractable otherwise.
- A core finite-valued language over a two-element set [17] is tractable if it is submodular (cf. Example 18) and intractable otherwise.
- A core finite-valued language over a three-element set [48] is intractable if it is submodular on a chain (cf. Example 19) or skew bisubmodular (cf. Example 23) and intractable otherwise.
- A finite-valued language containing all $\{0, 1\}$-valued unary cost functions [63] is tractable if it is submodular on a chain (cf. Example 24) and intractable otherwise.

Theorem 46 also implies a classification of the so-called Min-0-Ext problems [45].

For languages where the cost functions can take infinite values, no general complexity classification is known. In fact, even the special case of $\{0, \infty\}$-valued languages is a challenging open problem over sets with four or more elements as it corresponds to the complexity classification of the CSP (cf. Example 10). For the general VCSP, unlike the CSP, there is not even a well-established conjecture.

Nevertheless, some interesting and nontrivial partial results are known. For example, a complete complexity classification for valued constraint languages over a two-element set was established in [17]. Note that on a two-element set there is precisely one majority operation, as defined in Example 26, which we will denote by Mjrty, and precisely one minority operation, as defined in Example 27, which we will denote by Mnrty. There are also precisely two constant operations, which will be denoted Const_0 and Const_1.

[2] $\{0, 1\}$-valued languages correspond to Max-CSPs, cf. Example 12.

Theorem 47 (Classification of Boolean Languages [17])**.** *A valued constraint language* Γ *on* $D = \{0, 1\}$ *is tractable if it admits at least one of the following eight multimorphisms. Otherwise* $\mathrm{wRelClone}(\Gamma)$ *contains* ϕ_{nae} *or* ϕ_{xor} *and* Γ *is intractable.*

1. $\langle \mathrm{Const}_0 \rangle$
2. $\langle \mathrm{Const}_1 \rangle$
3. $\langle \mathrm{Min}, \mathrm{Min} \rangle$,
4. $\langle \mathrm{Max}, \mathrm{Max} \rangle$,
5. $\langle \mathrm{Min}, \mathrm{Max} \rangle$,
6. $\langle \mathrm{Mjrty}, \mathrm{Mjrty}, \mathrm{Mjrty} \rangle$,
7. $\langle \mathrm{Mnrty}, \mathrm{Mnrty}, \mathrm{Mnrty} \rangle$,
8. $\langle \mathrm{Mjrty}, \mathrm{Mjrty}, \mathrm{Mnrty} \rangle$.

Let us compare Theorem 47 with a classification of crisp Boolean languages, originally established by Schaefer in [77] and restated here using polymorphisms (see, e.g. [14]): A crisp constraint language on $D = \{0, 1\}$ is tractable if it admits one of the following six polymorphisms: Const_0, Const_1, Min, Max, Mjrty, Mnrty; otherwise it is intractable. These six tractable cases are covered by cases (1-4), (6), and (7) in Theorem 47. The six cases correspond to sets of Boolean relations that are 0-valid, or 1-valid, or expressible by Horn clauses, dual Horn clauses, 2-clauses, or linear equations over the field with 2 elements, respectively.

The hardness part of Theorem 47 can be rederived using the algebraic theory described in Section 4; see [24, 19] for details. We remark that if we restrict to core Boolean valued constraint languages, the first two cases in Theorem 47 disappear as those languages are not cores (and in fact are solvable trivially).

Another general complexity classification result concerns languages that contain all $\{0, 1\}$-valued unary cost functions. Note that a weighted polymorphism ω is called *conservative* if $f(x_1, \ldots, x_k) \in \{x_1, \ldots, x_k\}$ for all $f \in \mathrm{supp}(\omega)$.

Theorem 48 (Classification of Conservative Languages [63])**.** *Let* Γ *be a valued constraint language on a set* D *such that* Γ *contains all* $\{0, 1\}$*-valued unary cost functions on* D*. Then either* Γ *admits a conservative binary multimorphism* $\langle f_1, f_2 \rangle$ *and a conservative ternary multimorphism* $\left\langle f'_1, f'_2, f'_3 \right\rangle$ *and there is a family* M *of 2-element subsets of* D*, such that:*

- *for every* $\{a, b\} \in M$, $\langle f_1, f_2 \rangle$ *restricted to* $\{a, b\}$ *is a symmetric tournament pair (see Example 24), and*
- *for every* $\{a, b\} \notin M$, $\left\langle f'_1, f'_2, f'_3 \right\rangle$ *restricted to* $\{a, b\}$ *is an MJN multimorphism (see Example 28),*

in which case Γ *is tractable, or else* Γ *is intractable.*

The algorithm for solving the tractable case identified in Theorem 48 first enforces local consistency (see the discussion of bounded width at the beginning of this section). After this preprocessing step, any instance admits a symmetric tournament pair multimorphism [63] and is thus solvable using BLP.

We now briefly describe the partial classification results so far obtained for the Min-Cost-Hom and Min-Sol problems discussed in Examples 13 and 15 respectively. Recall that a Min-Cost-Hom problem corresponds to VCSP(Γ) for some language Γ containing only crisp cost functions and unary cost functions. Min-Sol problems are Min-Cost-Hom problems where the only unary cost function in Γ is a specific injective and finite-valued cost function.

The complexity classification for Min-Cost-Hom for languages containing all unary cost functions was established in [81]. The tractable case can be reduced, after a preprocessing step using local consistency techniques, to a certain problem on perfect graphs known to be solvable in polynomial time using linear programming [38]. For the special case of digraphs (i.e., when the only non-unary cost function allowed is a single binary crisp cost function), a complexity classification was obtained in [43].

The classification of Min-Cost-Hom for languages containing all unary crisp cost functions was initially studied in [80] and fully established in [85].

Finally, using the techniques from Section 4 and from [83], a very recent result has established the computational complexity of Min-Cost-Hom for all languages over a three-element set [86]. The only tractable cases either admit a weighted polymorphism with a semilattice operation in its positive support or a certain type of tournament pair. The former case is tractable using BLP by Theorem 44 and the latter case is tractable using a reduction to the result in [81] discussed above.

The classification of Min-Sol problems was established in [56] for maximal languages over a four-element set and for homogenenous languages. The classification of Min-Sol has recently also been obtained for all languages over a three-element set [85]. Using the notion of cores and the algebraic techniques from Section 4 and from [82, 83], three tractable cases have been identified: bisubmodular languages (Example 22), generalised min-closed languages (generalising Example 21), and generalised weak-tournament pair languages (generalising Example 24); the first two are solvable using BLP, by Theorem 44, while the last is solvable by a method similar to the tractable case from [81] discussed above.

Adapting the main result of [13] on CSPs, Powell and Krokhin have recently shown [74] that for every problem VCSP(Γ), where Γ is finite, there is a polynomial-time equivalent Min-Cost-Hom problem, VCSP(Γ'), where Γ' contains only a single crisp binary function and a single finite-valued unary function. Moreover, the equivalence also preserves (in both directions) many useful weighted polymorphisms of Γ, such as symmetric and weak near-unanimity polymorphisms [4]. Thus, in order to classify the computational complexity of *any* valued constraint

language it suffices to classify Min-Cost-Hom problems of this restricted form. This mirrors a similar reduction from the general CSP to the binary case which was first established in [29].

6 Approximation

Since many forms of valued constraint satisfaction problem are NP-hard, it is natural to study approximation algorithms for these problems, and their limits. Recall that a polynomial-time algorithm for an optimisation problem Π is called an r-approximation algorithm if, for each instance I of Π, the algorithm returns a solution S for I whose measure $m(S)$ satisfies the inequality

$$\max\left(\frac{m(S)}{OPT(S)}, \frac{OPT(S)}{m(S)}\right) \leq r.$$

The bound r is called the approximation ratio of the algorithm. Note that in general r can be a function of the size of I.

There has been major progress in the last 20 years in designing approximation algorithms and understanding the (in)approximability of combinatorial optimisation problems. The former direction was boosted by the application of techniques based on semidefinite programming (SDP) [34] whilst the latter was powered to a large extent by the theory of probabilistically checkable proofs, or PCPs, see [1]. A notable early source of inapproximability results is [40], where it is shown that certain problems (such as Max-3-Sat) can be approximated within a (problem-specific) constant r, but, unless P=NP, not within $r-\epsilon$ for any $\epsilon > 0$. There is now a large body of such optimal inapproximability results, including those for Minimum Vertex Cover and Max Cut, whose validity depends on the *Unique Games Conjecture*, or UGC (see survey [59]). This conjecture states that, for any $\epsilon > 0$, there is a large enough integer $k = k(\epsilon)$ such that it is NP-hard to distinguish two types of systems of linear equations of the form $x_i + x_j \equiv a_{ij} \pmod{k}$: those where at least a $(1-\epsilon)$-fraction of the equations can be satisfied and those where any assignment satisfies at most an ϵ-fraction of the equations. Despite the fact that the UGC has been used as a basis for many results, it is still open and the approximation community seems to be evenly divided as to which way it will eventually be resolved.

Semidefinite programming is an extension of linear programming where the variables are vectors in a high-dimensional space and the constraints, as well as the objective function, are linear in the inner products of these vectors. Any VCSP instance has a basic semidefinite programming relaxation similar to the BLP relaxation defined in Section 5. A breakthrough result of Raghavendra [75, 76] shows how to use the basic SDP relaxation to design, for *any* given finite and

finite-valued language Γ, an approximation algorithm for VCSP(Γ) that achieves some constant approximation ratio; moreover, this ratio cannot be improved unless the UGC is false. This ratio is not explicit, but there is an algorithm that can compute it with any given accuracy in doubly exponential time. It is interesting that this (conditionally) optimal ratio is related to a parameter of some objects similar to weighted polymorphisms. For more details, consult Raghavendra's paper and thesis [75, 76]; note that the (finite-valued) VCSP is referred to there as the generalized CSP or GCSP.

The class of all optimisation problems having a (polynomial-time) constant-factor approximation algorithm is denoted by APX. From the approximation point of view, the best type of algorithm is a PTAS (polynomial-time approximation scheme) which is actually a series of algorithms A_ϵ, $\epsilon > 0$, such that A_ϵ gives a $(1 + \epsilon)$-approximation and runs in time that is polynomial in the size of the instance (but not necessarily in $1/\epsilon$). One way to rule out the existence of a PTAS for a specific optimisation problem Π (unless P=NP) is to show that this problem is APX-hard, i.e., that every problem in APX has an approximation-preserving reduction to Π.

The classification results from Section 5 distinguish between (exact) polynomial solvability and NP-hardness. Some of these results can be strengthened to become dichotomies between polynomial solvability and APX-hardness For example, as discussed in Example 12, Max-CSP is equivalent to VCSP(Γ_{Max}) where Γ_{Max} consists of all cost functions taking only the values 0 and 1. For approximation results it is convenient to replace these with values with -1 and 0 respectively. Then the intractable cases of VCSP(Γ) with $\Gamma \subseteq \Gamma_{\mathsf{Max}}$ can be shown to be APX-hard (in fact, APX-complete, as each Max-CSP problem with a finite language belongs to APX) when Γ contains all unary $\{-1, 0\}$-valued functions [28] and when $|D| = 3$ [55].

There are only a few results concerning the approximability of VCSP(Γ) for languages Γ containing cost functions that can take infinite values. For example, it is shown in [44] that the problem VCSP($\{\phi_H\} \cup \Gamma^+_{\mathsf{unary}}$), a special case of Min-Cost-Hom (see Example 13) where $H = (V, E)$ is an undirected graph without loops and $\Gamma^+_{\mathsf{unary}}$ contains all unary functions with non-negative values, is not approximable within any factor if the List H-Colouring problem (cf. Example 11) is NP-complete and it has a $|V|$-approximation algorithm otherwise. As another example, the APX-hardness of some Min-Sol problems (Example 15) is established in [53].

7 The oracle model

In this paper we have assumed that the objective function in our problem is represented as a sum of functions each defined on some subset of the variables. There is a rich tradition in combinatorial optimisation of studying problems where the objective function is represented instead by a value-giving *oracle*. In this model a problem is tractable if it can be solved in polynomial time using only polynomially many queries to the oracle (where the polynomial is in the number of variables). Note that any query to the oracle can be simulated in linear time in the VCSP model. Hence, a tractability result (for a class of functions) in the oracle model automatically transfers to the VCSP model, while hardness results automatically transfer in the opposite direction.

One class of functions that has received particular attention in the oracle model is the class of submodular functions (cf. Example 18). There are several known algorithms for minimising a (finite-valued) submodular function using only a polynomial number of calls to a value-giving oracle (see [50, 51, 78]).

However, for some submodular valued constraint languages Γ, VCSP(Γ) can be solved much more efficiently than by using these general approaches. For example, the language Γ_{cut} defined in Example 8 can be solved in cubic time using the Min-Cut-based algorithm described in Example 8. A similar efficient approach can be used for all languages that are expressible over Γ_{cut}. However, it was shown in [88, 90] that not all submodular functions are expressible over Γ_{cut}, so this approach cannot be directly extended to solve arbitrary submodular VCSP instances. It is currently an open question whether the minimisation problem for submodular functions defined by sums of bounded arity submodular functions in the VCSP model is easier than general submodular function minimisation in the oracle model.

Other classes of finite-valued functions that can be efficiently minimised in the oracle model include bisubmodular and α-bisubmodular functions (Examples 22 and 23) [31, 71, 32, 47], functions with a 1-defect multimorphism (Example 25) [57], and functions that are submodular on certain lattices (Example 19) [64, 65]. The complexity of submodular function minimisation in the oracle model over arbitrary non-distributive lattices is still unknown (in the VCSP model, all such language are tractable, by Theorem 44).

The following general problem was mentioned in [48, 57, 82]: which weighted polymorphisms ω are sufficient to guarantee an efficient minimization algorithm, in the value-oracle model, for valued constraint languages Γ with $\omega \in \mathrm{wPol}(\Gamma)$? Natural candidates for which the question is open include the k-submodularity multimorphism for $k \geq 3$ from Example 22 and submodularity multimorphisms on many lattices from Example 19.

8 Conclusions and future directions

We have shown that the valued constraint satisfaction problem is a powerful general framework that can be used to express many standard combinatorial optimisation problems. The general problem is NP-hard, but there are many special cases that have been shown to be tractable. In particular, by considering restrictions on the cost functions we allow in problem instances, we have identified a range of different sets of cost functions that ensure tractability.

These restricted sets of cost functions are referred to as valued constraint languages, and we have described in Section 4 the very general algebraic techniques now being developed to classify the complexity of these languages.

This classification is still far from complete. In fact, even in the special case of the CSP (Example 10), where all cost functions take only the values 0 or ∞, there is still no complete classification of complexity for the corresponding constraint languages. This problem has been studied for many years, beginning with the seminal work of Feder and Vardi who conjectured that any such language will be either tractable or NP-complete [29]. This conjecture is still unresolved. However, the Algebraic Dichotomy conjecture [11] specifies the boundary between tractable and intractable languages, and it has been proved in many important cases. Naturally, it is desirable to develop the algebraic theory of VCSPs to the point where one could make a credible algebraic dichotomy conjecture for the VCSP, in order to have a specific target to aim at.

For finite-valued languages, the complexity classification is complete, see Theorem 46. One could ask, however, whether the tractability condition can be made tighter by being more specific about which binary symmetric weighted polymorphisms need to be present there. For $|D| = 2, 3$, tight descriptions are given in [17, 48].

The algebraic theory of the VCSP presented in Section 4 is based on the new notion of a weighted clone. Very little is known about weighted clones, and this direction is wide open for purely algebraic investigation. Some specific open problems include the (possible) description of weighted clones for $D = \{0, 1\}$, the identification of minimal weighted clones, and the investigation of classes of weighted clones supported by a given ordinary clone.

Further developing the algebraic theory of the VCSP using algebras and varieties [73] is a very promising direction of research because this theory works with a more general notion of expressibility. Possible algebraic dichotomy results from this theory would state that either a language expresses, in this more general way, a given function (usually with undesirable algorithmic properties of the corresponding VCSP) or else it has a "nice" weighted polymorphism. Such results [11, 67] have been fundamental to the success of the algebraic approach to complexity for the CSP.

It is natural to investigate how the operations that play a role in the algebraic theory for the CSP can be adapted to the VCSP setting. Examples of such conditions that we discussed earlier are weak near-unanimity and edge operations; there are several others. What can be said about valued constraint languages with weighted polymorphisms whose positive support includes such operations?

As we discussed in Section 5, only three algorithmic techniques seem to be sufficient to solve tractable crisp and finite-valued VCSPs (Bounded Width, Few Subpowers, and Basic LP relaxation). There also seem to be essentially only two seeds of hardness that cause intractability (NAE-like and XOR-like functions). Are there tractable general-valued VCSPs that require different techniques? Are there intractable general-valued VCSPs that can express neither NAE-like nor XOR-like functions?

The notion of weighted polymorphism works well for studying the exact solvability of the VCSP. It would be natural to explore its applicability to approximability questions for the VCSP and to oracle-tractability for classes of functions, as we discussed in Sections 6 and 7.

In this survey we have focused on the complexity of valued constraint satisfaction problems with restricted constraint languages. It is also possible to ensure tractability by restricting the structure of the constraint scopes - so-called *structural* restrictions [36, 37, 70]. Combining structural restrictions with language restrictions leads to so-called *hybrid* restrictions, and these provide a promising source of new tractable cases [21, 22] which has so far been very little explored.

References

[1] S. Arora and B. Barak. *Computational Complexity - A Modern Approach*. Cambridge University Press, 2009.

[2] L. Barto. The dichotomy for conservative constraint satisfaction problems revisited. In *LICS'11*, pages 301–310. IEEE Computer Society, 2011.

[3] L. Barto and M. Kozik. Absorbing subalgebras, cyclic terms and the constraint satisfaction problem. *Logical Methods in Computer Science*, 8, 2012.

[4] L. Barto and M. Kozik. Constraint Satisfaction Problems Solvable by Local Consistency Methods. *Journal of the ACM*, 61(1), 2014. Article No. 3.

[5] F. Börner. Basics of Galois connections. In *Complexity of Constraints*, volume 5250 of *LNCS*, pages 38–67. Springer, 2008.

[6] Y. Boykov, O. Veksler, and R. Zabih. Markov random fields with efficient approximations. In *Computer Vision and Pattern Recognition*, pages 648–655. IEEE Computer Society, 1998.

[7] A. Bulatov. A dichotomy theorem for constraint satisfaction problems on a 3-element set. *Journal of the ACM*, 53(1):66–120, 2006.

[8] A. Bulatov. Bounded relational width. Manuscript, 2009.

[9] A. Bulatov and V. Dalmau. A simple algorithm for Mal'tsev constraints. *SIAM Journal on Computing*, 36(1):16–27, 2006.

[10] A. Bulatov and P. Jeavons. Algebraic structures in combinatorial problems. Technical Report MATH-AL-4-2001, Technische Universität Dresden, 2001.

[11] A. Bulatov, A. Krokhin, and P. Jeavons. Classifying the Complexity of Constraints using Finite Algebras. *SIAM Journal on Computing*, 34(3):720–742, 2005.

[12] A. A. Bulatov. Complexity of conservative constraint satisfaction problems. *ACM Transactions on Computational Logic*, 12(4), 2011. Article 24.

[13] J. Bulín, D. Delic, M. Jackson, and T. Niven. On the Reduction of the CSP Dichotomy Conjecture to Digraphs. In *CP'13*, volume 8124 of *LNCS*, pages 184–199. Springer, 2013.

[14] D. Cohen and P. Jeavons. The complexity of constraint languages. In F. Rossi, P. van Beek, and T. Walsh, editors, *The Handbook of Constraint Programming*. Elsevier, 2006.

[15] D. Cohen, M. Cooper, P. Jeavons, and A. Krokhin. Supermodular Functions and the Complexity of MAX-CSP. *Discrete Applied Mathematics*, 149(1-3):53–72, 2005.

[16] D. A. Cohen, M. C. Cooper, and P. G. Jeavons. An Algebraic Characterisation of Complexity for Valued Constraints. In *CP'06*, volume 4204 of *LNCS*, pages 107–121. Springer, 2006a.

[17] D. A. Cohen, M. C. Cooper, P. G. Jeavons, and A. A. Krokhin. The Complexity of Soft Constraint Satisfaction. *Artificial Intelligence*, 170(11):983–1016, 2006b.

[18] D. A. Cohen, M. C. Cooper, and P. G. Jeavons. Generalising submodularity and Horn clauses: Tractable optimization problems defined by tournament pair multimorphisms. *Theoretical Computer Science*, 401(1-3):36–51, 2008.

[19] D. A. Cohen, M. C. Cooper, P. Creed, P. Jeavons, and S. Živný. An algebraic theory of complexity for discrete optimisation. *SIAM Journal on Computing*, 42(5):915–1939, 2013.

[20] M. C. Cooper, S. de Givry, M. Sánchez, T. Schiex, M. Zytnicki, and T. Werner. Soft arc consistency revisited. *Artificial Intelligence*, 174(7–8):449–478, 2010.

[21] M. C. Cooper and S. Živný. Hybrid tractability of valued constraint problems. *Artificial Intelligence*, 175(9-10):1555–1569, 2011.

[22] M. C. Cooper and S. Živný. Tractable triangles and cross-free convexity in discrete optimisation. *Journal of Artificial Intelligence Research*, 44:455–490, 2012.

[23] Y. Crama and P. L. Hammer. *Boolean Functions - Theory, Algorithms, and Applications*. Cambridge University Press, 2011.

[24] P. Creed and S. Živný. On minimal weighted clones. In *CP'11*, volume 6876 of *LNCS*, pages 210–224. Springer, 2011.

[25] N. Creignou, S. Khanna, and M. Sudan. *Complexity Classification of Boolean Constraint Satisfaction Problems*, volume 7 of *SIAM Monographs on Discrete Mathematics and Applications*. SIAM, 2001.

[26] N. Creignou, P. G. Kolaitis, and H. Vollmer, editors. *Complexity of Constraints: An Overview of Current Research Themes*, volume 5250 of *LNCS*, 2008. Springer.

[27] E. Dahlhaus, D. Johnson, C. Papadimitriou, P. Seymour, and M. Yannakakis. The Complexity of Multiterminal Cuts. *SIAM Journal on Computing*, 23(4):864–894, 1994.

[28] V. Deineko, P. Jonsson, M. Klasson, and A. Krokhin. The approximability of Max CSP with fixed-value constraints. *Journal of the ACM*, 55(4), 2008. Article 16.

[29] T. Feder and M. Y. Vardi. The Computational Structure of Monotone Monadic SNP and Constraint Satisfaction: A Study through Datalog and Group Theory. *SIAM Journal on Computing*, 28(1):57–104, 1998.

[30] S. Fujishige. *Submodular Functions and Optimization*, volume 58 of *Annals of Discrete Mathematics*. North-Holland, Amsterdam, 2nd edition, 2005.

[31] S. Fujishige and S. Iwata. Bisubmodular Function Minimization. *SIAM Journal on Discrete Mathematics*, 19(4):1065–1073, 2005.

[32] S. Fujishige, S. Tanigawa, and Y. Yoshida. Generalized skew bisubmodularity: A characterization and a min-max theorem. Technical Report RIMS-1781, Kyoto University, 2013.

[33] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W.H. Freeman, 1979.

[34] M. X. Goemans and D. P. Williamson. Improved approximation algorithms for maximum cut and satisfiability problems using semidefinite programming. *Journal of the ACM*, 42(6):1115–1145, 1995.

[35] A. V. Goldberg and R. E. Tarjan. A New Approach to the Maximum Flow Problem. *Journal of the ACM*, 35(4):921–940, 1988.

[36] G. Gottlob, G. Greco, and F. Scarcello. Tractable Optimization Problems through Hypergraph-Based Structural Restrictions. In *ICALP'09*, volume 5556 of *LNCS*, pages 16–30. Springer, 2009.

[37] M. Grohe. The complexity of homomorphism and constraint satisfaction problems seen from the other side. *Journal of the ACM*, 54(1):1–24, 2007.

[38] M. Grötschel, L. Lovasz, and A. Schrijver. *Geometric Algorithms and Combinatorial Optimization*, volume 2 of *Algorithms and Combinatorics*. Springer, 1988.

[39] G. Gutin, P. Hell, A. Rafiey, and A. Yeo. A dichotomy for minimum cost graph homomorphisms. *European Journal of Combinatorics*, 29(4):900–911, 2008.

[40] J. Håstad. Some optimal inapproximability results. *Journal of the ACM*, 48(4): 798–859, 2001.

[41] P. Hell and J. Nešetřil. *Graphs and Homomorphisms*. Oxford University Press, 2004.

[42] P. Hell and J. Nešetřil. Colouring, constraint satisfaction, and complexity. *Computer Science Review*, 2(3):143–163, 2008.

[43] P. Hell and A. Rafiey. The Dichotomy of Minimum Cost Homomorphism Problems for Digraphs. *SIAM Journal on Discrete Mathematics*, 26(4):1597–1608, 2012.

[44] P. Hell, M. Mastrolilli, M. M. Nevisi, and A. Rafiey. Approximation of Minimum Cost Homomorphisms. In *Proceddings of the 20th Annual European Symposium on Algorithms (ESA12)*, pages 587–598, 2012.

[45] H. Hirai. Discrete Convexity and Polynomial Solvability in Minimum 0-Extension Problems. In *SODA'13*, pages 1770–1778. SIAM, 2013.

[46] A. Huber and V. Kolmogorov. Towards minimizing k-submodular functions. Technical Report arXiv:1309.5469, 2013.

[47] A. Huber and A. Krokhin. Oracle tractability of skew bisubmodular functions. Technical Report arXiv:1308.6505, 2013.

[48] A. Huber, A. Krokhin, and R. Powell. Skew bisubmodularity and valued CSPs. *SIAM Journal on Computing*, 2014. To appear.

[49] P. M. Idziak, P. Markovic, R. McKenzie, M. Valeriote, and R. Willard. Tractability and learnability arising from algebras with few subpowers. *SIAM Journal on Computing*, 39(7):3023–3037, 2010.

[50] S. Iwata. Submodular Function Minimization. *Mathematical Programming*, 112(1): 45–64, 2008.

[51] S. Iwata and J. B. Orlin. A Simple Combinatorial Algorithm for Submodular Function Minimization. In *SODA'09*, pages 1230–1237, 2009.

[52] P. G. Jeavons, D. A. Cohen, and M. Gyssens. Closure Properties of Constraints. *Journal of the ACM*, 44(4):527–548, 1997.

[53] P. Jonsson and G. Nordh. Introduction to the MAXIMUM SOLUTION Problem. In *Complexity of Constraints*, volume 5250 of *LNCS*, pages 255–282. Springer, 2008.

[54] P. Jonsson and J. Thapper. Approximability of the maximum solution problem for certain families of algebras. In *CSR'09*, volume 5675 of *LNCS*, pages 215–226. Springer, 2009.

[55] P. Jonsson, M. Klasson, and A. Krokhin. The Approximability of Three-valued MAX CSP. *SIAM Journal on Computing*, 35(6):1329–1349, 2006.

[56] P. Jonsson, F. Kuivinen, and G. Nordh. MAX ONES Generalized to Larger Domains. *SIAM Journal on Computing*, 38(1):329–365, 2008.

[57] P. Jonsson, F. Kuivinen, and J. Thapper. Min CSP on Four Elements: Moving Beyond Submodularity. In *CP'11*, volume 6876 of *LNCS*, pages 438–453. Springer, 2011.

[58] S. Khanna, M. Sudan, L. Trevisan, and D. Williamson. The approximability of constraint satisfaction problems. *SIAM Journal on Computing*, 30(6):1863–1920, 2001.

[59] S. Khot. On the Unique Games Conjecture (Invited Survey). In *CCC'10*, pages 99–121. IEEE Computer Society, 2010.

[60] V. Kolmogorov. Submodularity on a tree: Unifying $L^\natural$-convex and bisubmodular functions. In *MFCS'11*, volume 6907 of *LNCS*, pages 400–411. Springer, 2011.

[61] V. Kolmogorov. The power of linear programming for finite-valued CSPs: a constructive characterization. In *ICALP'13*, volume 7965 of *LNCS*, pages 625–636. Springer, 2013.

[62] V. Kolmogorov, J. Thapper, and S. Živný. The power of linear programming for general-valued CSPs. 2013. arXiv:1311.4219.

[63] V. Kolmogorov and S. Živný. The complexity of conservative valued CSPs. *Journal of the ACM*, 60(2), 2013. Article No. 10.

[64] A. Krokhin and B. Larose. Maximizing Supermodular Functions on Product Lattices, with Application to Maximum Constraint Satisfaction. *SIAM Journal on Discrete Mathematics*, 22(1):312–328, 2008.

[65] F. Kuivinen. On the complexity of submodular function minimisation on diamonds. *Discrete Optimization*, 8(3):459–477, 2011.

[66] R. Ladner. On the Structure of Polynomial Time Reducibility. *Journal of the ACM*, 22:155–171, 1975.

[67] B. Larose and P. Tesson. Universal algebra and hardness results for constraint satisfaction problems. *Theoretical Computer Science*, 410(18):1629–1647, 2009.

[68] S. L. Lauritzen. *Graphical Models*. Oxford University Press, 1996.

[69] M. Maróti and R. McKenzie. Existence theorems for weakly symmetric operations. *Algebra Universalis*, 59(3-4):463–489, 2008.

[70] D. Marx. Tractable hypergraph properties for constraint satisfaction and conjunctive queries. *Journal of the ACM*, 60(6), 2013. Article No. 42.

[71] S. T. McCormick and S. Fujishige. Strongly polynomial and fully combinatorial algorithms for bisubmodular function minimization. *Mathematical Programming*, 122(1):87–120, 2010.

[72] M. Mezard and A. Montanari. *Information, Physics, and Computation*. Oxford University Press, 2009.

[73] J. Ochremiak. Algebraic properties of valued constraint satisfaction problem. Technical report, 2014. arXiv:1403.0476.

[74] R. Powell and A. Krokhin. A reduction of VCSP to digraphs. Manuscript, 2014.

[75] P. Raghavendra. Optimal algorithms and inapproximability results for every CSP? In *STOC'08*, pages 245–254. ACM, 2008.

[76] P. Raghavendra. *Approximating NP-hard problems: Efficient algorithms and their limits*. PhD thesis, University of Washington, 2009.

[77] T. J. Schaefer. The Complexity of Satisfiability Problems. In *STOC'78*, pages 216–226. ACM, 1978.

[78] A. Schrijver. *Combinatorial Optimization: Polyhedra and Efficiency*, volume 24 of *Algorithms and Combinatorics*. Springer, 2003.

[79] A. Szendrei. *Clones in Universal Algebra*, volume 99 of *Seminaires de Mathematiques Superieures*. University of Montreal, 1986.

[80] R. Takhanov. Extensions of the Minimum Cost Homomorphism Problem. In *COCOON'10*, volume 6196 of *LNCS*, pages 328–337. Springer, 2010a.

[81] R. Takhanov. A Dichotomy Theorem for the General Minimum Cost Homomorphism Problem. In *STACS'10*, pages 657–668, 2010b.

[82] J. Thapper and S. Živný. The power of linear programming for valued CSPs. In *FOCS'12*, pages 669–678. IEEE, 2012.

[83] J. Thapper and S. Živný. The complexity of finite-valued CSPs. In *STOC'13*, pages 695–704. ACM, 2013.

[84] D. Topkis. *Supermodularity and Complementarity*. Princeton University Press, 1998.

[85] H. Uppman. The Complexity of Three-Element Min-Sol and Conservative Min-Cost-Hom. In *ICALP'13*, volume 7965 of *LNCS*, pages 804–815. Springer, 2013.

[86] H. Uppman. Computational Complexity of the Extended Minimum Cost Homomorphism Problem on Three-Element Domains. In *STACS'14*, volume 25, pages 651–662, 2014.

[87] M. J. Wainwright and M. I. Jordan. Graphical models, exponential families, and variational inference. *Foundations and Trends in Machine Learning*, 1(1-2):1–305, 2008.

[88] S. Živný. *The Complexity and Expressive Power of Valued Constraints*. PhD thesis, University of Oxford, 2009.

[89] S. Živný. *The complexity of valued constraint satisfaction problems*. Cognitive Technologies. Springer, 2012. ISBN 978-3-642-33973-8.

[90] S. Živný, D. A. Cohen, and P. G. Jeavons. The Expressive Power of Binary Submodular Functions. *Discrete Applied Mathematics*, 157(15):3347–3358, 2009.